Introducing FxCop

Jeffrey van Gogh
Michael Murray

June 2004

Internal Technical Education

What Is FxCop?

- A free-for-download, stand-alone code analysis tool that examines managed assemblies for design and code correctness issues
- A console and graphical application that manages analysis targets, rule sets, and messages

- Goals
  - Support adoption of Microsoft® .NET Design Guidelines
  - Transfer expert knowledge regarding technical issues and potential "gotchya"s
  - Establish best practices that minimize code defects and maintenance cost

demo

How Does It Work?

What Can FxCop Rules Do?

- Access all managed metadata
- Examine IL method bodies
- Walk assembly call graphs
- Determine some argument values for call sites
- Use the spelling checker (if Microsoft Office is installed)

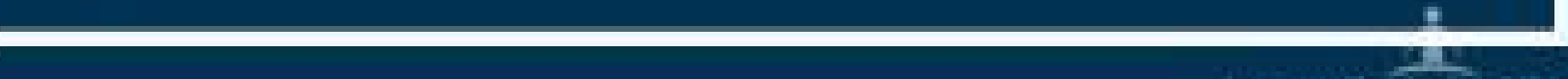
Types of FxCop Rules

- COM Interop Rules: Detect COM interop issues
- Design Rules: Detect potential design flaws
- Globalization Rules: Detect missing or incorrect usage of information related to globalization and localization
- Naming Rules: Regard casing, keyword collisions, and other issues around public members
- Performance Rules: Detect patterns that will affect or degrade performance
- Usage Rules: Detect potential flaws in the way you use Fx or .NET APIs.
- Security Rules: Detect programming elements in your assemblies that leave your assemblies vulnerable to malicious users or code (this is where we'll spend our time for the rest of the presentation)
- Custom Rules: Your own FxCop rules

FxCop COM Rules

- Rules that support integrating .NET applications and binaries with COM clients
- Examples:
- [ComVisible(true)]
  public class ComCreatable
  {
  private ComCreatable()
  public static ComCreatable Factory()
  { ..}
  }

- [ComVisible(true)]
  public struct AccountInfo
  {
  private string m_SecurityCode;
  public string Name;
  }



FxCop Design Rules

- Rules that support good library design
- Examples:

```
public class UserInfo
{
    private string m_password;
    public string Name { get {..} set {..} }
    public string Password { set {..}}

    SaveUserInfo(FileStream stream)
    {
        using(StreamWriter writer = new StreamWriter(stream))
        {
                writer.WriteLine("Name: " + Name);
                writer.WriteLine("Password:" + m_password);
        }
    }
}
```

FxCop Globalization Rules

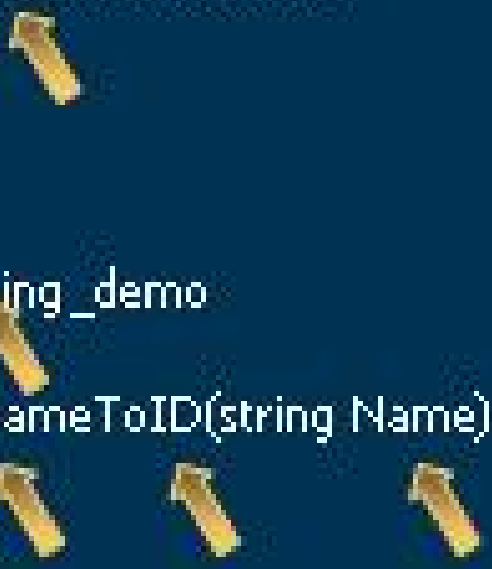
- Rules that support world-ready libraries and applications

```
Public void SaveToXml(double version)
{
    if (version < 1.0) throw new
      ArgumentException("Unsupported version: " +
      version.ToString());

  XmlWriter writer = new XmlWriter(
                "C:\program files\MyApp\config.xml");

  writer.WriteElementString("Version",
      version.ToString());
}
```

FxCop Naming Rules

- Rules that support the naming conventions from the .NET Design Guidelines
- Examples:

```
Public class MyError : Exception
{
    ...
}

Public class naming_demo
{
    public string nameToID(string Name)
    {
        ...
    }
}
```



FxCop Performance Rules

- Rules that support improved performance
- Examples:

```
Public string PretttyPrintNumbers(int[] list)
{
    string output == "";
    for(int i = 0; i < list.Length; i++)
    {
        if (output != "") output +=", ";
        output += list[i].ToString();
    }
    //not needed anymore
    // return PrettyPrintHelper(output);
    return output;
}
Private string PrettyPrintHelper(string input) { ... }
```

FxCop Security Rules

- Examples of security rules:

```
[PermissionSet (SecurityAction.LinkDemand, Name="FullTrust")]
Public class SecurityInfo
{
   public readonly byte[] Key = GenerateKey();
}
Public struct DangerousInformation
{
   [PermissionSet (SecurityAction.Demand, Name="FullTrust")]
   public DangerousInformation(string owner)
   {...}
}
```

FxCop Usage Rules

- Rules that support proper usage of the .NET Framework

```
Public class UsageDemo
{
    public void EvaluateArguments(int value, string data)
    {
            if (value < 3)
                    throw new ArgumentException("value",
            "must be above 3");
            if (data == null)
                    throw new ArgumentNullException(
                            "required value", "data");
    }

    public MyCollection Items
    {
            get { ... }
            set { ... }
    }
}
```

demo

New Capabilities in 1.30

- Better Microsoft Visual Studio® Side-by-Side Experience
- Resolved Missing Dependencies
- Analysis of Any Version of an Assembly
- UI Improvements
- Analysis Improvements
- New Rules

Integrating FxCop into Your Development Process

- Using fxcopcmd and XML files
- Integrating into your development process
  - Use command-line and baseline everything
  - Run as checkin system and start catching new violations, fix, then address baseline
  - You should fix all Design Rule fixes—and focus on consistency

demo

Creating Custom Rules

The Future:
Visual Studio Team System

- FxCop built directly into Visual Studio
- Superset of stand-alone FxCop rules
- New integrated environment for all analysis
- Analysis can be enabled as build step
- Messages, help topics appear in IDE

Issues with FxCop—FYI

- FxCop is about guidelines not hard and fast rules
- False positives
- Lack of integration with Visual Studio
- Maintaining FxCop XML files across developer projects

(Rude) Questions About FxCop

- Questions
  - Why do Microsoft .NET Framework libraries generate violations?
  - Why does FxCop generate violations?
  - Why not ship FxCop source code?
  - Will there continue to be a free version of FxCop once VSTS ships?
  - Others?...

Community Resources

FxCop on GotDotNet
http://www.gotdotnet.com/team/fxcop/

FxCop Team Weblog
http://blogs.msdn.com/fxcop/

CLR Base Class Libraries (BCL) Team Site
http://www.gotdotnet.com/team/clr/bcl/default.aspx

Brad Abrams' Weblog—Design Guidelines
http://blogs.msdn.com/brada/

Rico Mariani's Weblog—.NET Performance
http://blogs.msdn.com/ricom/